ARMY RESEARCH LABORATORY

An Analysis of CyberSleuth Traffic in a Tactical Internet Environment

by Binh Q. Nguyen

ARL-TR-3108 October 2003

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

Army Research Laboratory
Adelphi, MD 20783-1197

ARL-TR-3108 **October 2003**

An Analysis of CyberSleuth Traffic in a Tactical Internet Environment

Binh Q. Nguyen
Computational and Information Sciences Directorate, ARL

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
October 2003

2. REPORT TYPE
Technical Report

3. DATES COVERED (From - To)

4. TITLE AND SUBTITLE
An Analysis of CyberSleuth Traffic in a Tactical Internet Environment

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Binh Q. Nguyen

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
ATTN: AMSRL-CI-CN
2800 Powder Mill Road
Adelphi, MD 20783-1197

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-3108

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Research Laboratory
2800 Powder Mill Road
Adelphi, Maryland 20783-1197

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Preliminary results of an analysis of CyberSleuth traffic data are reported in this document. CyberSleuth is a prototype of an adaptive agent-based vulnerability system intended for digitized tactical information systems. The analysis focuses on identifying the size of the mobile agents used in CyberSleuth and the required time for a mobile agent to travel between two networked computers. The traffic data were captured in an experiment conducted in the Tactical Internet laboratory of the U.S. Army Communications-Electronic Research, Development, and Engineering Center (CERDEC) in August 2002. The knowledge learned from studying the analyzed results could potentially lead to a better understanding of the constrained behavior of a very low bandwidth communication environment in which a potential application of mobile agents would have to operate. CyberSleuth is being refined for possible transition to CERDEC under the Technical Program Annex number CE-CI-1999-10.

15. SUBJECT TERMS
Mobile agent, tactical internet, vulnerability assessment

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	UL	41	Binh Q. Nguyen
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(Include area code)* (301) 394-1199

Contents

List of Figures

Preface

Initial results of an analysis of the traffic data of CyberSleuth captured in the Tactical Internet laboratory of the U.S. Army Communications-Electronics Research, Development, and Engineering Center (CERDEC) are reported in this document. CyberSleuth is a prototype of an adaptive agent-based vulnerability system being developed by the U.S Army Research Laboratory for potential transition to CERDEC under the Technical Program Annex number CE-CI-1999-10.

The concept of CyberSleuth was originally developed by the consortium team members of the Advanced Telecommunications and Information Distribution Research Program, operated under the Federated Laboratory Program from 1996 to 2001.

Acknowledgments

The author would like to thank everyone who was involved in the development and testing of CyberSleuth in the Tactical Internet (TI) laboratory, especially P. Syckle, D. Chen, and R. Genke of the U.S. Army Communications and Electronics Command (CECOM), M. Little of Telcordia, and M. Gaughan of BAE Systems. P. Syckle supplied the needed resources and organizational supports for conducting the experiment. D. Chen managed the TI laboratory and provided insightful knowledge of the TI equipment. R. Genke ensured that all the equipment was operable and captured the CyberSleuth traffic data. M. Little designed the architectural framework of CyberSleuth, and M. Gaughan implemented the concept using the Java programming language.

INTENTIONALLY LEFT BLANK.

Executive Summary

Preliminary results of an analysis of CyberSleuth traffic data are reported in this document. CyberSleuth is a prototype of an adaptive agent-based vulnerability system that can operate in two modes: paranoid mode and suspicious mode. The paranoid mode enables continuous and random assessments of system vulnerability at target hosts. The suspicious mode uses the assessment results performed in the paranoid mode as a basis for concurrent and autonomous assessments. The operational concept of CyberSleuth was originally developed and demonstrated by the consortium team members of the Advanced Telecommunications and Information Distribution Research Program (*1*), which was part of the Federated Laboratory Program, 1996-2001.

The traffic data were captured in an experiment conducted from 13 to 21 August 2002 in the Tactical Internet (TI) laboratory of the U.S. Army Communications-Electronic Research, Development, and Engineering Center (CERDEC), Fort Monmouth, NJ. Only the performance of CyberSleuth operating in the paranoid mode was observed in the experiment. The suspicious mode of CyberSleuth was inoperable because there was no version of the Amzi! software that could run on Solaris™ x86, the operating system used in the TI. The experiment employed a computer to serve as a mobile agent dispatcher capable of sending assessment agents to two other computers by two radio links via single-channel ground-air radio system (SINCGARS) and enhanced position location and reporting system (EPLRS) tactical radios.

The analysis focuses on identifying the size of the mobile agents used in CyberSleuth and the required time for a mobile agent to travel between two networked computers because these two properties affect the performance of CyberSleuth and influence the design of size-efficient mobile agents potentially appropriate for the resource-constrained environment. The effective throughputs, calculated with the measured sizes and transmission time for CyberSleuth traffic on the EPLRS radio link were relatively consistent at about 400 bits per second (bps). On the other hand, the calculated throughputs for CyberSleuth traffic on the SINCGARS radio link varied widely from about 200 bps to 1,578 bps. Concurrent communication sessions and frequent transmissions of short messages could possibly cause this variation.

The knowledge learned from the empirical results could potentially lead to a better understanding of the constrained behavior of a very low bandwidth communication environment in which a potential application of mobile agents would have to operate. The limited bandwidth available for CyberSleuth in the experiment was a difficult environment in which the mobile assessment agents operated. The size of the mobile assessment agents was a concern because it proportionally affected the transmission time. Future challenges for CyberSleuth developers will include reducing the size of the mobile agents without sacrificing their effectiveness and conducting further empirical studies to corroborate several claimed features of CyberSleuth, including but not limited to its adaptivity and efficiency.

INTENTIONALLY LEFT BLANK.

1. Introduction

1.1 Background

CyberSleuth is a prototype of an adaptive agent-based vulnerability system whose concept was originally developed and demonstrated by the consortium team members of the Advanced Telecommunications and Information Distribution Research Program (*1*), which was part of the Federated Laboratory Program from 1996 to 2001. The CyberSleuth team members consist of the U.S. Army Research Laboratory (ARL), the City College of the City University of New York, Telcordia Technologies, and BAE[*] SYSTEMS. CyberSleuth is now further refined for potential transition to the Space and Terrestrial Communications Directorate of the U.S. Army Communications-Electronic Research, Development, and Engineering Center, which is also known as the U.S. Army Communications and Electronics Command, under the Technical Program Annex (TPA) number CE-CI-1999-10.

CyberSleuth is designed for potential operation in tactical digitized information networks that have constrained computational resources and limited bandwidth. These computing environments can accommodate only incremental assessments, which are performed by the mobile assessment agents (MA) of CyberSleuth running in either paranoid mode or suspicious mode. Assessments are continuously and randomly performed in the paranoid mode and are concurrently and autonomously performed in the suspicious mode. The MA technology is a relatively novel network-computing paradigm with many features that have been claimed to be suitable for low bandwidth networks (*2*, *3*, *4*). Five significant functional components of CyberSleuth are depicted in figure 1.

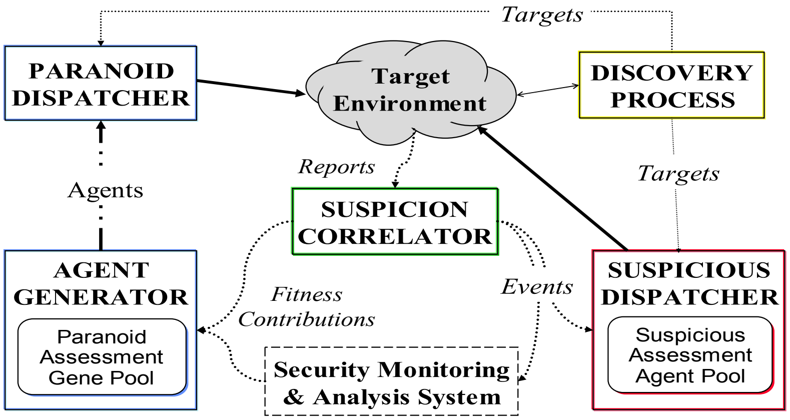

Figure 1. Functional architecture (*1*).

[*] British Aerospace

The discovery process (DP) component, situated on the upper right corner, explores the underlying network to find potential computing targets that are capable of hosting the execution of the mobile assessment agents of CyberSleuth. The DP also classifies the assessment targets on the basis of their hardware architecture and operating systems, thereby automatically obtaining topographic information of the network for subsequent use in assessment processes. For this reason, the DP runs at the beginning of the execution of CyberSleuth.

The two components on the left, the paranoid dispatcher and the agent generator, function in tandem to perform assessments in the paranoid mode, which operates on the assumption that a known vulnerability exists somewhere in the network. The agent generator dynamically and adaptively creates assessment agents suitable for target hosts. The paranoid dispatcher launches mobile assessment agents to target hosts and receives them as they return from having completed an assessment at a remote host.

In the center and on the lower right corner of figure 1 are the suspicious correlator and the suspicious dispatcher that also operate in tandem to direct assessments in the suspicious mode, which runs on a clear evidence that a vulnerability exists in a specific host. The suspicious correlator analyzes the received assessment results and decides whether the suspicious dispatcher should dispatch a mobile agent equipped with a set of comprehensive and relevant assessment techniques for a particular type of vulnerability.

Enabling technologies used in CyberSleuth consist of Java[1] technologies and a custom Amzi![2] inference engine software. Java technologies include a mobile-agent execution environment called Aglets[3] and a JavaBeans[4] component architecture. All mobile assessment agents are built with Aglets technology, and they are designed to carry various JavaBeans components for execution at a target host. Each JavaBeans component implements a mechanism for assessing a specific vulnerability. The Amzi! inference engine is used to develop the suspicious correlator capable of analyzing and evaluating assessment results to determine their significance and implication. Complete functionality of CyberSleuth therefore theoretically could be demonstrated on any computing platform that runs a suitable Java virtual machine and an appropriate Amzi! inference engine.

1.2 Scope and Purposes

This document reports preliminary results of an analysis of CyberSleuth traffic data that were captured in an experiment conducted as part of the Tactical Command and Control (C2) Protect Advanced Technological Demonstration (ATD) program in a laboratory environment of a tactical digitized communications and information network, the Tactical Internet (TI) laboratory of CERDEC. The analysis neither evaluates the overall performance of CyberSleuth nor determines its usefulness or appropriateness in the TI environment.

The analysis focuses on identifying the size of the mobile agents used in CyberSleuth and the required time for a mobile agent to travel between two networked computers because these two properties affect the performance of CyberSleuth and influence the design of size-efficient

[1] The source for Java technology, URL: http://java.sun.com.

[2] Prolog software development environment and inference engine, URL: http://www.amzi.com/.

[3] Aglets software development kit, URL: http://aglets.sourceforge.net/.

[4] The JavaBeans Component Architecture, URL: http://java.sun.com/products/javabeans/.

mobile agents potentially appropriate for a resource-constrained environment, such as the TI which is being used by the 4th Infantry Division, the first digital force of the U.S. Army. TI provides a communications link from the foxhole to the Pentagon; it is a hierarchical network of digital computers that use various tactical radios as a means for transporting digital information over the air medium at various bandwidths. Low bandwidths are available at the low-level echelons, and higher bandwidths are available at higher echelons.

The intended audience of this report includes technical personnel and management involved in the design and development of network-centric applications for TI.

The next section describes the TI environment and the testing method that was used in the experiment. The Results section documents the results obtained in the experiment. The Discussion section discusses the results reported in the previous section. The Conclusions and Recommendations section provides a summary of substantiated findings, together with their implication for the development of MA applications and a list of recommendations for implementation and future research opportunities. Appendix A provides a list of acronyms that were used in this report. Appendix B includes the Python program that was used to parse the log data. Appendix C gives a sample of the log data.

2. Method

The experiment was conducted in the TI laboratory of CERDEC from 13 to 21 August 2002. Each collaborating host was preconfigured with the Force XXI Battle Command Brigade and Below **(**FBCB2**)** software[5] running in a homogeneous environment of Sun Solaris™ x86 operating systems. The FBCB2, working together with tactical radios, provides two-dimensional graphical situation awareness (SA) and digital C2 services to the land warrior. The experiment employs three computers in the TI laboratory, as depicted in figure 2.

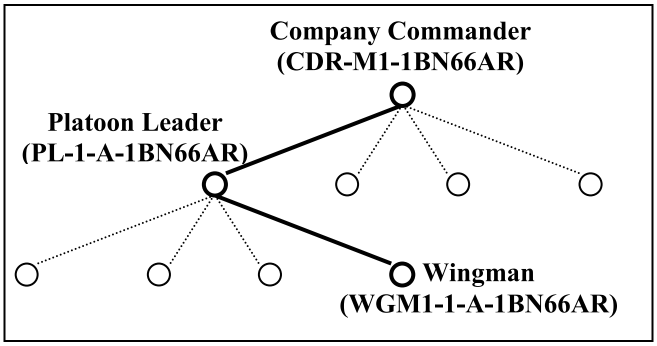

Figure 2. Experiment setup.

[5] The FBCB2 Software Information Center,
URL: https://peoc3s-res.monmouth.army.mil/QuickPlace/fbcb2_software/main.nsf.

Each computer was then additionally loaded with supporting software that CyberSleuth needed to function properly: an appropriate Java virtual machine and an Aglet execution environment suitable for dispatching and running mobile assessment agents. Because of limited network bandwidth availability, the size of each agent was configured to carry two assessment mechanisms capable of checking two known vulnerabilities at each target host although theoretically, an assessment agent could carry as many mechanisms as the system resource could support. The CyberSleuth software was loaded on only one computer, the one that would belong to a platoon leader in reality. From this computer, CyberSleuth dispatched assessment agents to two other computers: the Wingman and the company commander computers. A pair of two enhanced position location and reporting system (EPLRS)[6] radios served as the digital data communication link between the platoon leader and the company commander. A pair of single-channel ground-to-air radio system (SINCGARS)[7] provided the digital and voice communication between the platoon leader and the Wingman; voice communications would take precedence over digital communication (*5*) but were not used during the experiment to provide CyberSleuth an uninterrupted operation.

Only the performance of CyberSleuth operating in the paranoid mode was shown in the TI laboratory; the discovery process and the ability of CyberSleuth to operate in the suspicious mode were not demonstrable because of a concern about the limited network bandwidth availability and the lack of a version of the Amzi! software that could run on Solaris™ x86 operating systems. The discovery process was disabled because of a concern about its execution time, which would have prolonged the experiment because it would have had to contact every networked computer with its name listed in the host table and to determine the availability of a suitable mobile-agent execution environment in which assessment agents would run. Therefore, the topographic information of the three participating networked computers was manually entered in the host table. The suspicious mode of CyberSleuth was entirely inactive during the experiment because the Amzi! inference engine was unavailable.

The traffic data generated by CyberSleuth operating in the paranoid mode were captured and stored in two text files. Information obtained from the traffic data files includes the names of the initiator and the responder, the starting and ending times of each communication session, and number of transmitted data bytes. Analyzing these data files requires the development of a computer program that can parse, decipher, reorganize, and tabulate the dense and cryptic material into a format that could be imported into the Microsoft Excel™ program for subsequent computation and plotting. The computer program was written in the Python programming language and is included in appendix B.

3. Results

Effective throughput, the number of data bits transmitted per second, and the sizes of assessment agents were calculated with the information contained in the two data files—the files *TCP226a* and *CS63aTCP*. The file *TCP226a* has approximately 72 minutes worth of traffic data traveled

[6] Description of EPLRS, URL: http://www.monmouth.army.mil/ peoc3s/trcs/EPLRSDescr.htm.

[7] Description of SINCGARS, http:/www.monmouth.army.mil/ peoc3s/trcs/GarsDesc.htm.

between the platoon leader (PL) computer and the Wingman (WGM) over the SINCGARS link and between the PL and the commander (CDR) computer over the EPLRS link. The file *CS63aTCP* has more than 1,000 minutes worth of traffic data traveled only between the PL and the WGM computers over the SINCGARS radio link.

Information obtained and derived from the file *TCP226a* is displayed in figures 3 through 7, showing the sizes and the calculated throughput of data chunks traveled between the PL and the CDR computers and between the PL and the WGM computers. Similarly, information obtained and derived from the file *CS63aTCP* is depicted in figures 8 through 12, showing the sizes and the calculated throughput of data chunks traveled between the PL and the WGM computers.

A consistent pattern of mobile-agent traffic data traveled between the PL and the CDR computers captured in the file *TCP226a* is plotted in figure 3, showing the starting time of each communication session. The total size of the mobile assessment agents dispatched from the PL computer was slightly less than that of the returning agents. During this 72-minute run, the total size of the dispatched agents was 60,939 bytes and that of the returning agents was 65,672 bytes. The increased size of the returning agents was attributable to the additional storage that was allocated by the system to accommodate the assessment results. The figure also shows that for each communication session, the receiver steadily sent to the initiator 93 bytes of data (iPL<=rCDR and iCDR<=rPL).

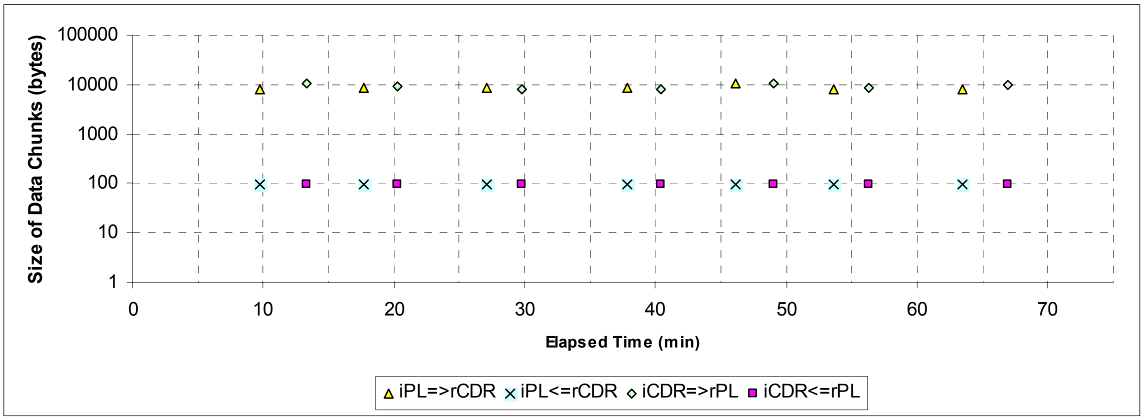

Figure 3. Sizes of data chunks transmitted on the EPLRS link.

A mixed traffic pattern of mobile-agent traffic data traveled between the PL and the WGM computers captured in the file *TCP226a* is plotted in figure 4. For the size of data chunks that are larger than 10,000 bytes, a consistent pattern of 93-byte data returned to the initiator by the receiver is observed; furthermore, figure 4 reveals that the WGM computer initiated several requests for transmissions of many smaller chunks of data from the PL computer between the transmissions of the large chunks of data. This phenomenon appears to be that the mobile-agent execution environment at the target host did not have the required Java classes for a complete execution of the dispatched mobile agent; therefore, it had to connect with the PL dispatcher to request additional Java classes. This process occurred automatically without any user intervention.

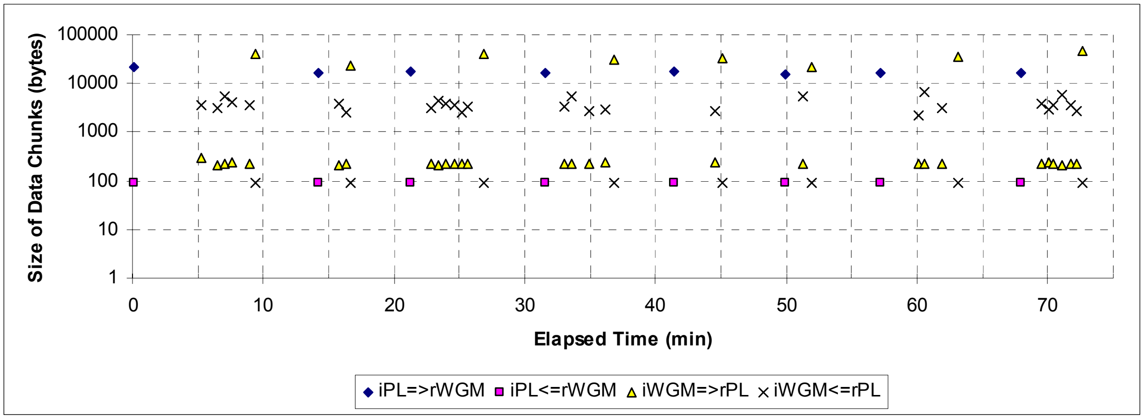

Figure 4. Sizes of data chunks transmitted on the SINCGARS link.

Again, the total size of the mobile assessment agents dispatched from the PL computer was less than that of the returning agents. During this 72-minute run, the total number of bytes of the data traveling from the dispatcher PL to the target computer WGM was 243,043 bytes and that of the data traveling from the target computer WGM to the dispatcher was 280,483 bytes.

The throughput was calculated and depicted in figures 5 and 6. The throughput for the assessment agents traveling between the PL computer and the CDR computer over a pair of EPLRS radios were consistently slightly below 500 bits per second (bps), and the sizes of these mobile agents were clustered around 10,000 bytes. On the other hand, the throughput for the assessment agents traveling between the PL computer and the WGM computer over the SINCGARS radio link varied widely, especially for the agents that traveled from the WGM computer to the PL computer (iWGM=>rPL).

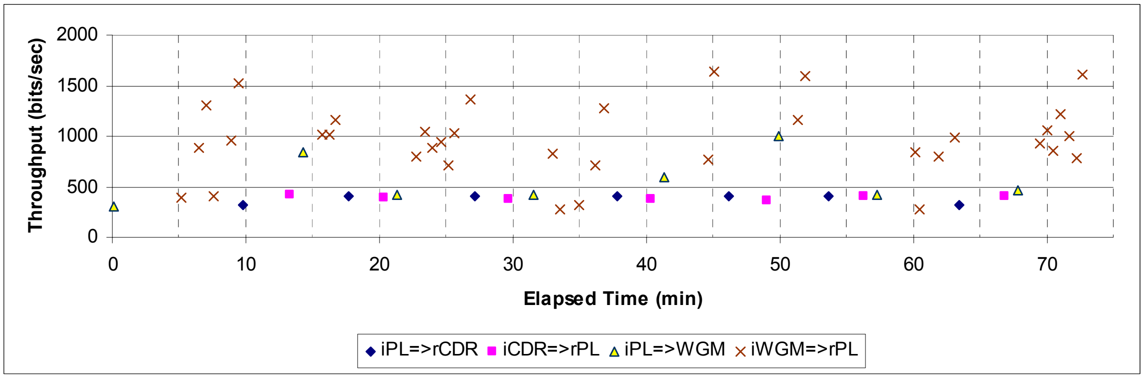

Figure 5. Throughput for transmitted data versus time.

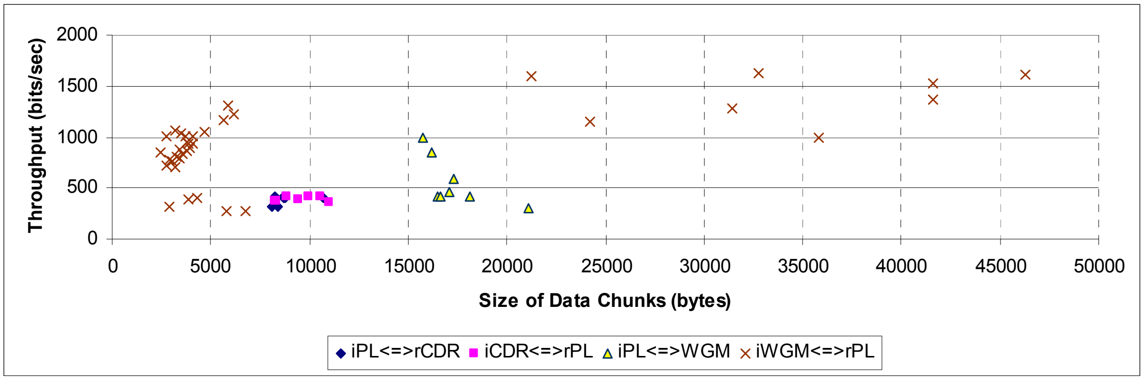

Figure 6. Throughput versus sizes of data chunks.

During the first 17 minutes, concurrent and overlapped communication sessions originated from the initiators to the receivers and were extracted from the file *TCP226a*; they are shown in figure 7. For example, while an assessment agent was traveling from the PL computer to the WGM computer [iPL=>rWGM (1)], the WGM computer completed five transmissions of short communication messages with the PL computer [iWGM=>rPL (1-5)] and started an overlapped communication session [iWGM=>rPL (*6*)]. Meanwhile, having completed a session with the WGM computer, the PL computer began transmitting data to the CDR computer [iPL=>rCDR], and when this session ended, the PL computer received transmitted data from the CDR computer [iCDR=>rPL] and simultaneously sent data to the WGM computer [iPL=>rWGM(2)].

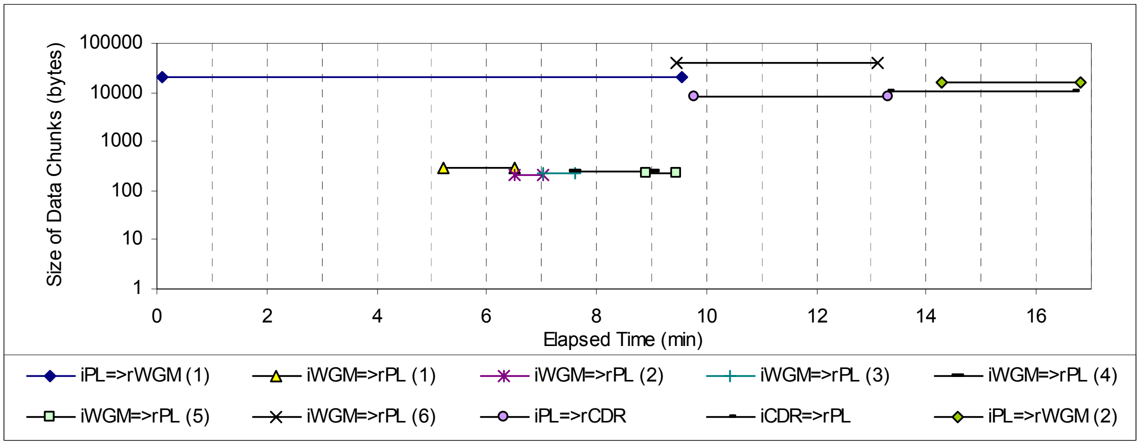

Figure 7. A sample of concurrent transmission.

Information obtained and derived from the second file, *CS63aTCP*, is displayed in figures 8 through 12. Figure 8 shows the sizes of data chunks traveled between the PL and the WGM computers over the SINCGARS link over a period of about 1200 minutes. The sizes of the data transmitted (returning agents) from the WGM computer to the PL computer (iWGM=>rPL) were expectedly larger than those of the data transmitted (dispatched agents) from the PL computer to the WGM computer (iPL=>rWGM). Several small sizes of communication messages that originated from the WGM computer to the PL computer (iWGM=>rPL) are clustered in the beginning of the plot.

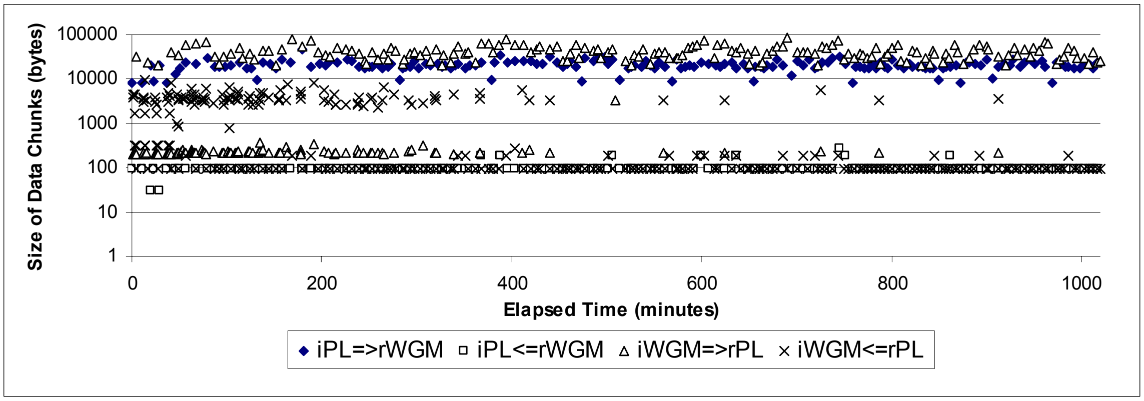

Figure 8. Sizes of data chunks transmitted over the SINCGARS link.

An expanded plot of figure 8 during the first 50 minutes is plotted in figure 9, showing that the WGM computer often initiated several short communication messages to the PL computer, which in turn responded with data chunks that were larger than the requesting messages. This phenomenon appears to indicate that the WGM computer was sending requests to the dispatcher, the PL computer, for additional Java code to complete the execution of the dispatched agents at their hosts.

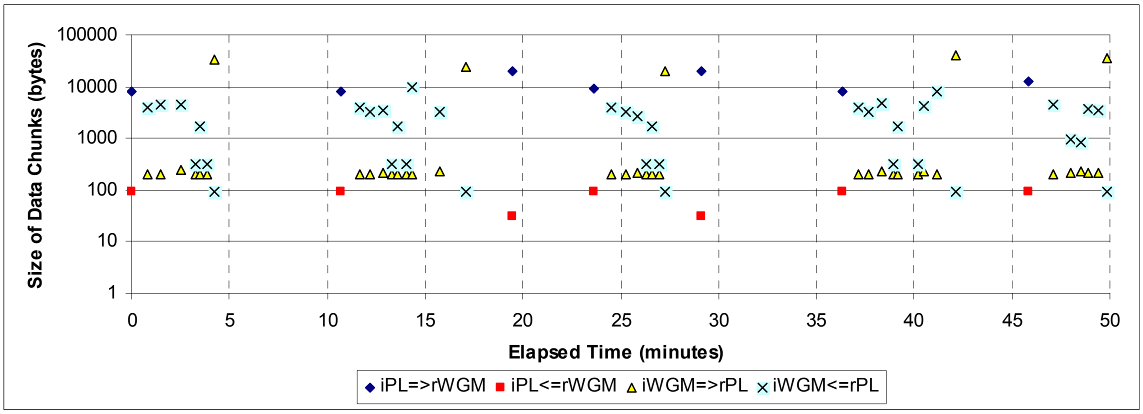

Figure 9. Sizes of data chunks—first 50 minutes.

The computed effective throughput, plotted against elapsed time and the sizes of the data chunks are shown in figures 10 and 11. Figure 10 shows the calculated throughput versus the elapsed times. Most of the throughput for the data transmitted from the PL computer to the WGM computer (iPL=>rWGM) was between 300 bps and about 800 bps, whereas, the throughput for the data transmitted from the WGM computer to the PL computer (iWGM=>rPL) varied widely because of several short communication messages that originated from the WGM computer (iWGM=>rPL), as plotted in figure 9.

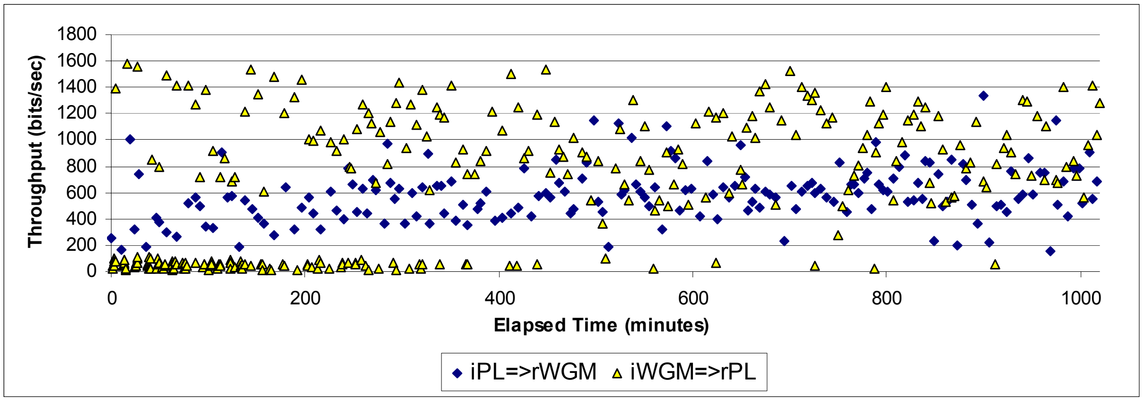

Figure 10. Throughput versus time.

The throughput shown in figure 11 were calculated by dividing the total number of transmitting and responding data bytes by their transmission times. Once again, the throughput for the data traveling from the WGM computer to the PL computer over the SINCGARS link varied widely; however, the throughput generally increased proportionally to the size of the data transmitted.

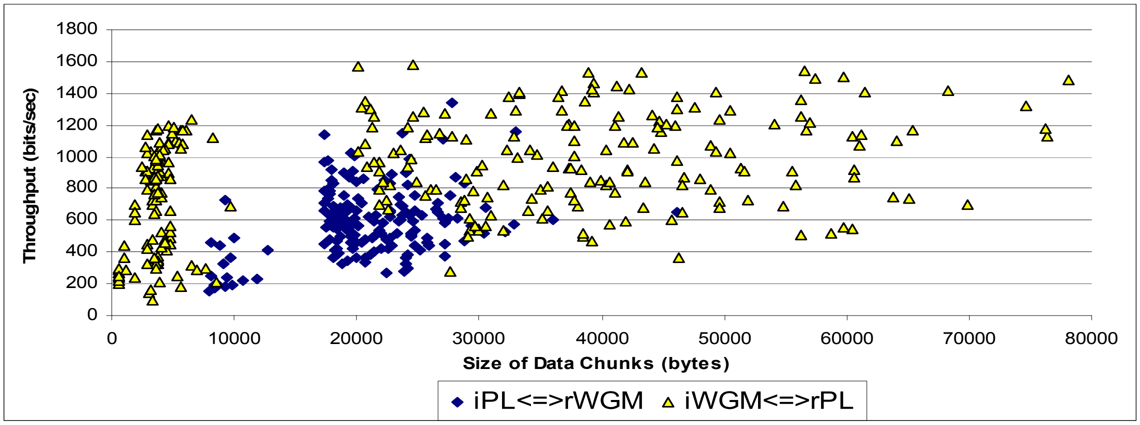

Figure 11. Throughputs versus sizes of data chunks.

Information about the concurrent and overlapped transmission of CyberSleuth traffic was also available in the log file *CS63aTCP*. The activities that occurred during the first 8 minutes of run are depicted in figure 12. For example, an assessment agent required 4.29 minutes to travel from the PL computer to the WGM computer (iPL=>rWGM). During this time, the WGM computer sent 12 short communication transactions with the PL computer (iWGM=>rPL) and initiated an overlapped communication session that lasted for 3.19 minutes (from 4.20 to 7.39 minutes).

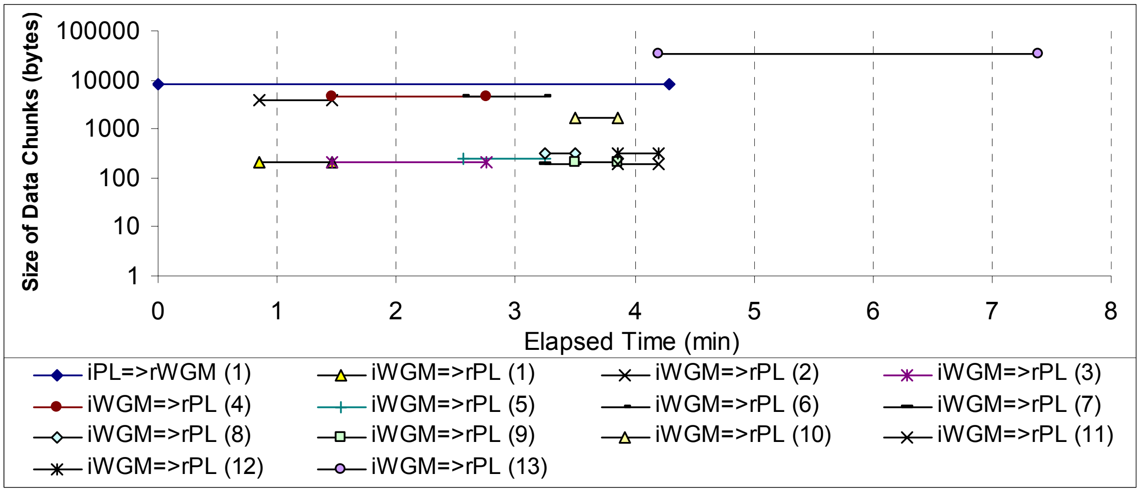

Figure 12. Sample of concurrent and overlapped transmissions.

4. Discussion

Limited communication bandwidth available on the TI environment was a challenge for CyberSleuth to operate. A dispatched mobile assessment agent requires nearly 5 minutes to travel between two hosts, taking approximately 10 minutes for a round trip, assuming that the assessment time at the host is negligible, compared to the travel time. To minimize the required travel time for mobile assessment agents, each dispatched agent used in the demonstration was configured to carry only two assessment mechanisms capable of inspecting and evaluating two known system-configuration weaknesses and errors. CyberSleuth would require approximately 24 hours to examine 300 specific known vulnerabilities that could exist in a single target.

The calculated throughput for CyberSleuth traffic on the EPLRS radio link between the PL computer and the CDR computer were relatively consistent at about 400 bps. On the other hand, the calculated throughput for CyberSleuth traffic on the SINCGARS radio link varied widely from about 200 bps to 1,578 bps. This variation could be caused by concurrent communication sessions and frequent transmissions of short messages, which required the same overhead communication data bytes for establishing and disconnecting a communication session, and thus reduced the overall effective throughput.

These results were obtained from a very simple network configuration involving only three computers that used two pairs of different tactical radio links. Had the PL computer been connected to four WGM computers connected by SINCGARS radio links operating at the same frequency, then the throughput would have been substantially reduced because of network contention and congestion. Had CyberSleuth been deployed at the CDR computer, then the throughputs would have further deteriorated because every dispatched agent to a WGM computer would have had to travel through two different radio links, taking two "hops" to arrive and return.

5. Conclusions and Recommendations

Preliminary results of an analysis of the data traffic have been presented and discussed. The empirical data, created by the mobile assessment agents of CyberSleuth operating in a TI environment, form the basis for this analysis. The knowledge learned from the analyzed results could potentially lead to a better understanding of the constrained behavior of a very low bandwidth communication environment in which a potential application of mobile agents would have to operate.

The limited bandwidth available for CyberSleuth in the experiment was a difficult environment in which the mobile assessment agents operated. The size of the mobile assessment agents was a concern because it proportionally affected the transmission time. Future challenges for CyberSleuth developers will include reducing the size of the mobile agents without sacrificing their effectiveness and conducting further empirical studies to corroborate several claimed features of CyberSleuth, including but not limited to its adaptivity and efficiency.

The implementation of each assessment mechanism as a JavaBean™ component provides modular software components and a random combination of which would enable an assessment agent to assess different vulnerabilities each time it visits a target host. Combining several components would not only increase the effectiveness of the dispatched assessment agents but would also enlarge their sizes, thereby raising their round trip travel time. To increase the efficiency without sacrificing the effectiveness of the mobile assessment agent per trip, implementing more than one assessment mechanism in a JavaBean™ component is recommended.

6. References

[1] Little, M.; Gaughan, M.; Ferrari, G.; Tardif, A.; Conner, M.; Cirincione, G.; Younger, M.; and Tzatzalos, C. “CyberSleuth: an Adaptive Agent-based Vulnerability Assessment System for Military Networks,” *Advanced Telecommunications and Information Distribution Progam (ATIRP) Final Report 1996-2001*, pp. 5.1-5.25, U.S. Army Research Laboratory, Adelphi, MD, June 2001.

[2] Lange, B.; and Oshima, M. Seven Good Reasons for Mobile Agents, *Communications of the ACM*, March 1999**.**

[3] Chess, D.; Harrison, C.; and Kershenbaum, A. "Mobile Agents: Are They a Good Idea?” *IBM Research Division, T.J. Watson Research Center*, Yorktown Heights, New York, 12/2194 - Declassified 3/16/95. URL: http://citeseer.nj.nec.com/chess95mobile.html

[4] Gray, R. “Soldiers, Agents and Wireless Networks: A Report on a Military Application,” Proceedings of the Fifth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents, Manchester, England, April, 2000.

[5] Headquarters, Department of the Army, “Field Manual 11-32 Combat Net Radio Operation,” U.S. Army Signal Center and Fort Gordon (ATTN: ATZH-DTL), Fort Gordon, GA 30905, 10 February 1990.

Appendix A. Acronyms

ARL	U.S. Army Research Laboratory
ATD	Advanced Technology Demonstration
C2	command and control
CDR	(company) commander
CECOM	The U.S. Army Communications and Electronics Command
CERDEC	The U.S. Army Communications-Electronic Research, Development, & Engineering
DP	discovery process
EPLRS	enhanced position location and reporting system
FBCB2	Force XXI Battle Command Brigade and Below
MA	mobile agent
PL	platoon leader
SA	situation awareness
SINCGARS	single-channel ground-to-air radio system
TI	Tactical Internet
TPA	technical program annex
WGM	Wingman

INTENTIONALLY LEFT BLANK.

Appendix B. Computer Code Used in the Analysis

```
"""
Analyzing the two data files captured in the TI Lab of CECOM.
    'TCP226a.txt'
    'CS63aTCP.txt'
                                        --- bnguyen@arl, Spring 2003.


Usage: import analyze
    or
    simply double click this file, the file "analyze.py"
"""


import string, os
class TCPIPinfo:
    def __init__(self, fn=None):
        print 'TCPIPinfo::__init(fn)__', fn
        self.ioprefix   = fn[:fn.find('.txt')]
        self.iofile_ext = fn[fn.find('.txt'):]
        self.iofilename = self.ioprefix + '-info' + self.iofile_ext
        self.ioplotfile = self.ioprefix+'-plotduration'+self.iofile_ext
        self.outfile    = open(self.iofilename, 'w')
        self.plotfile   = open(self.ioplotfile, 'w')
        self.time0          = 0 # constant throughout the life of an object.
        self.reset()

    def reset(self):
        self.source  = ''
    self.sink    = ''
    self.stime   = ''
    self.etime   = ''
    self.xtime   = 0
    self.size    = 0
    self.r2isize = 0
    self.throughput=0

def set_source(self, ip):
    """set the source to be the name of the initiator."""
    self.source = self.ip2name(ip)

def set_sink(self, ip):
    """set the destination to be the name of the responder."""
    self.sink = self.ip2name(ip)

def set_start_time(self, t):
    """ set the start time of each TCP/IP session
        (time(i)-time0)= elapsed time from the beginning of a run.
    """
    self.stime   = t
    if self.time0 <= 0 : # the beginning of a run.
        self.time0 = self.to_msec(t)

def set_end_time(self, t):
    """set etime = end time of a TCP/IP session"""
    self.etime   = t

def set_xtime(self):
    """set the transmission time (duration) of a TCP/IP session"""
```

```
        self.xtime  = self.to_msec(self.etime)-self.to_msec(self.stime)
        #
        # session went over midnight
        # 00:00:00.000 <= t <= 23:59:59.999
        # 24:00:00.000 = 23:59:59.999 + 1/1000
        if self.xtime < 0 :
            self.xtime = (1000*3600*24) - \
                self.to_msec(self.stime) + self.to_msec(self.etime)

    def add_size(self, s):
        """Tabulating the number of bytes transferred from the initiator to the responder"""
        self.size = self.size + s

    def add_r2isize(self, s):
        """tabulating the number of bytes that the responder sent to the initiator"""
        self.r2isize = self.r2isize + s

    def cal_throughput(self): # bits/sec (8 bits=byte, 1000 msec=sec)
        """ calculate the throughput for the current session"""
        self.throughput = 8*1000*float(self.size)/float(self.xtime)

    def end_processing(self):
        self.outfile.close()
        self.plotfile.close()
        self.separate_traffic()
        print 'End of Processing - Results are saved in the file: <%s>'
                        % self.iofilename

    def separate_traffic(self):
        exec_file_name = self.ioprefix + '-exe'
        of = open(exec_file_name, 'w')
        of.write('grep CDR %s > %s-CDR.txt\n' % (self.iofilename,
                        self.ioprefix))
        of.write('grep WGM %s > %s-WGM.txt\n' % (self.iofilename,
                        self.ioprefix))
        of.write('grep    ^WGM %s-WGM.txt > %s-WGM2PL.txt\n' %
                        (self.ioprefix, self.ioprefix))
        of.write('grep -v ^WGM %s-WGM.txt > %s-PL2WGM.txt\n' %
                        (self.ioprefix, self.ioprefix))
        of.write('grep    ^"  PL" %s-CDR.txt > %s-PL2CDR.txt\n' %
                        (self.ioprefix, self.ioprefix))
        of.write('grep -v ^"  PL" %s-CDR.txt > %s-CDR2PL.txt\n' %
                        (self.ioprefix, self.ioprefix))
        of.write('cat %s-*2*.txt >> %s-traffic.txt\n' % (self.ioprefix,
                        self.ioprefix))
        of.close()
        print 'separate_traffic() - DONE!'
        print 'chmod():', os.system('chmod +w %s' % exec_file_name)
        print 'exec() :', os.system(exec_file_name)

    def output(self):
        if len(self.source) <= 0 :
            print 'TCPIPinfo::output() - nothing to print.'
            return

        if self.size <= 0 and self.r2isize <= 0 :
            print '%16s, %16s, %s, %s : No data tranferred --> no
                        output.' % (self.source, self.sink, self.stime,
                        self.etime)
            return
```

```
elapsed_time = (self.to_msec(self.stime)-self.time0)/60000.0
if elapsed_time < 0 :
    elapsed_time = elapsed_time + 24*60

self.outfile.write('%16s, %16s, %s, %s, %d, %d, %5.2f, %8d,
                %5.2f, %6d, %6d, %10.3f\n' %  \
        (self.source, self.sink, self.stime, self.etime,     \
         self.to_msec(self.stime), self.to_msec(self.etime), \
         elapsed_time, self.xtime, self.xtime/60000.0,       \
         self.size, self.r2isize, self.throughput))
"""
Prepare data for plotting the duration using MS Excel.
Example: o--------o      |-------|    x------x
src, dst, t(s)-t0, src2dst(bytes)
src, dst, t(e)-t0, src2dst(bytes)
"""
ts = (self.to_msec(self.stime)-self.time0)/60000.0
te = (self.to_msec(self.etime)-self.time0)/60000.0
if ts < 0 : ts = ts + 24*60
if te < 0 : te = te + 24*60
self.plotfile.write('%16s, %16s, %5.2f, %d\n' % (self.source,
                self.sink, ts, self.size))
self.plotfile.write('%16s, %16s, %5.2f, %d\n' % (self.source,
                self.sink, te, self.size))
self.plotfile.write('%16s, %16s, %5.2f, %d\n' % (self.source,
                self.sink, ts, self.r2isize))
self.plotfile.write('%16s, %16s, %5.2f, %d\n' % (self.source,
                self.sink, te, self.r2isize))
```

```
def ip2name(self, ip):
    """
        converts the given IP address to a host/unit name.
        xxx.xxx.xxx.xxx ----> CDR-M1-1BN66AR
        yyy.yyy.yyy.yyy ----> PL-1-A-1BN66AR
        zzz.zzz.zzz.zzz ----> WGM1-1-A-1BN66AR
    """

    if ip == 'xxx.xxx.xxx.xxx'   :
        return 'CDR-M1-1BN66AR'
    elif ip == ‘yyy.yyy.yyy.yyy' :
        return 'PL-1-A-1BN66AR'
    elif ip == 'zzz.zzz.zzz.zzz’:
        return 'WGM1-1-A-1BN66AR'
    else :
        return ip #nothing was converted.
        #return '***error***'

def to_msec(self, t):
    """
        Convert time XX:XX:XX.mmm to milliseconds.
        00:00:00.000 <= t <= 23:59:59.999
    """
    i=string.find(t, ':')
    hrs = long(t[0:i])
    i += 1
    j=string.find(t, ':', i)
    min = long(t[i:j])
    j += 1
    i=string.find(t, '.', j)
    sec = long(t[j:i])
    i += 1
    mil = long(t[i:])

    return(mil + 1000*(sec + 60*min + 3600*hrs))

########################################################################
# Looking for overlapped TCP/IP sessions.
#
def print_overlapped_sessions(self):
    FIELD_SEPARATOR=','
    OUT_TXT_FILENAME = self.ioprefix + '-overlaps'+self.iofile_ext

    i = open(self.iofilename, 'r')
    contents = i.readlines()
    i.close()

    start_times = {}
    end_times = {}
    count       = 0

    print 'Reading input file'
    for input_line in contents :
        #
        # save the start time and end time in two lists
        #
        fields = input_line.split(FIELD_SEPARATOR)
        start_times[count]= self.to_msec(fields(2))
        end_times[count]  = self.to_msec(fields(3))
        count = count + 1
```

```
print 'Finished reading input file.'

print 'Adjust recorded time ...'
one_full_day = 24 * 3600 * 1000 #milliseconds.
count = 0
for i in range(1,len(start_times)) :
    if start_times[i]< start_times[0] or \
       end_times[i]  < start_times[0] :
            start_times[i] = start_times[i] + one_full_day
            end_times[i]   = end_times[i]   + one_full_day
print 'End adjusting times.'
#
# outputing the results.
#
out_txt_file = open(OUT_TXT_FILENAME, 'w')
concurrent_sessions = []
for i in range(len(start_times)) :
    #
    # Assume that times are recorded in time-ascending order
    #
    overlapped_sessions=[]
    for j in range(i+1, len(start_times)):
        if start_times[j] < end_times[i]:
            #if another TCP/IP session is started while
            #   the current one is in progress. Use a list
            # to store the indices of overlapped sessions
            #occurred in the session i
            overlapped_sessions.append(j)
            if j not in concurrent_sessions :
                    concurrent_sessions.append(j)
                #end if

        nOverlaps = len(overlapped_sessions)
        if nOverlaps == 0 and i not in concurrent_sessions :
            #no interruption
            out_txt_file.write('<%d:--no overlapped sessions>'% i)
            out_txt_file.write('  (%d)%s' % (i, contents[i]))
        elif nOverlaps == 1 :
            #output the indices of the overlapped sessions...
            out_txt_file.write('<%d: %d>\n' % (i,
                              overlapped_sessions[0] ) )
            out_txt_file.write('  (%d)%s' % (i, contents[i]))
            #output the contents
out_txt_file.write('  (%d)%s' %(overlapped_sessions[0], contents[overlapped_sessions[0]]))
        elif nOverlaps > 0 :
            #output the indices of the overlapped sessions...
            out_txt_file.write('<%d:' % i)
            for j in range(nOverlaps-1) :
                out_txt_file.write(' %d,' % overlapped_sessions[j])
            out_txt_file.write(' %d>\n' % overlapped_sessions[j+1] )
            out_txt_file.write('  (%d)%s' % (i, contents[i]))
            #output the contents
            for j in range(nOverlaps) :
                out_txt_file.write('  (%d)%s' %
(overlapped_sessions[j],contents[overlapped_sessions[j]]))
    out_txt_file.close()
    print 'print_overlapped_sessions()  : <%s>' % OUT_TXT_FILENAME
```

```
##############################################################################

class ParseTCPIP:
    #
    # Parsing the TCP/IP data captured in the TI lab in August, 2002.
    #
    def ffxtract(self, str, s, e):
        """ extract a substring from str, given delimiters s & e  """
        i = str.find(s)        #looking for 's' from the left.
        j = str.find(e, i+1)   #looking for 'e' from s+1.
        return str[i+1:j]

    def frfxtract(self, str, s, e):
        """ extract a substring from str, given delimiters s & e  """
        i = str.find(s)    #looking for 's' from the left.
        j = str.rfind(e)   #looking for 'e' from the right.
        return str[i+1:j]

    def analyze(self, fn):
        transmitter = 'Initiator:'
        receiver    = 'Responder:'
        label       = 'Time'
        underline   = '----'
        trans2rec   = 'I -> R'
        rec2trans   = 'I <- R'
        bytes       = 'bytes'

        session_start = 0
        start_time_next = 0 #next input line contains starting time.
        previous_line = ''
```

```
result = TCPIPinfo(fn)
#read the input data file.
infile = open(fn);
while 1 :
    input_line = infile.readline()
    if len(input_line) == 0 :
        break
    input_line = string.strip(input_line)

    if input_line.find(transmitter) >= 0 :
        session_start = 1 #extracting the ip addr of the source
        result.set_source(self.frfxtract(input_line,' ',':'))
        continue
    if session_start <= 0 :
        continue    #searching for the 'transmitter' keyword
    if input_line.find(receiver) >= 0 :
        #extracting the ip address of the receiver
        result.set_sink(self.frfxtract(input_line,' ',':'))
    elif input_line.find(label) >= 0 :
        continue    #ignore label
    elif input_line.find(underline) >= 0 :
        start_time_next = 1 #next input line will be data.
        continue
    elif len(input_line) > 1:
        if start_time_next >= 1 :   #1st transmission.
            result.set_start_time(self.ffxtract(input_line,' ',' '))
            start_time_next = 0     #ignore other time data
        else :   # Not 1st transmission, which is usually a SYNC flag.
            if input_line.find(trans2rec) >= 1 and \
                input_line.find(bytes) >= 1 : #extract N bytes
```

```
                  size = string.atoi(self.frfxtract(input_line,'(',' '))
                  result.add_size(size)
               elif input_line.find(rec2trans) >= 1 and \
                  input_line.find(bytes) >= 1 : #extract N bytes
                  size = string.atoi(self.frfxtract(input_line,'(',' '))
                  result.add_r2isize(size)
            previous_line = input_line   #save the current input line
         else :
            session_start = 0   #end of session.  save the end time
            result.set_end_time(self.ffxtract(previous_line,' ',' '))
            result.set_xtime()          # calculate transmission time
            result.cal_throughput()     # number of bits per second
            result.output()
            result.reset()

      infile.close()
      result.end_processing()
      result.print_overlapped_sessions()

#########################################################
"""
Usage: import analyze
   or
   simply double click this file, the file "analyze.py"
"""
ParseTCPIP().analyze('TCP226a.txt')
ParseTCPIP().analyze('CS63aTCP.txt')
```

Appendix C: A Sample of Traffic Data

```
Initiator: xxx.xxx.xxx.xxx:---- (port IIII)

Responder: yyy.yyy.yyy.yyy:---- (port JJJJ)

Time             Tap Dir  Msg Dir  Seq Num    Ack Num    Flags

---------------- -------  -------  ---------- ---------- ----------

226 17:53:10.966   T      I -> R   S:9EA3A6C3 A:00000000 Syn

226 17:53:53.463   T      I -> R   S:9EA3A6C3 A:00000000 Syn

226 17:53:55.094    R     I <- R   S:381A0272 A:9EA3A6C4 Syn Ack

226 17:53:55.116   T      I -> R   S:9EA3A6C4 A:381A0273 Ack

226 17:53:55.128   T      I -> R   S:9EA3A6C4 A:381A0273 Ack Psh (536 bytes)

226 17:54:40.109   T      I -> R   S:9EA3A6C4 A:381A0273 Ack (536 bytes)

226 17:54:47.024    R     I <- R   S:381A0273 A:9EA3A6C4 Ack

226 17:54:47.040    R     I <- R   S:381A0273 A:9EA3A8DC Ack

226 17:54:47.056    R     I <- R   S:381A0273 A:9EA3A8DC Ack

226 17:54:47.058   T      I -> R   S:9EA3A8DC A:381A0273 Ack (536 bytes)

226 17:54:47.210   T      I -> R   S:9EA3AAF4 A:381A0273 Ack Psh (536 bytes)

226 17:54:55.670    R     I <- R   S:381A0273 A:9EA3AAF4 Ack

226 17:54:55.686    R     I <- R   S:381A0273 A:9EA3AD0C Ack

226 17:54:55.687   T      I -> R   S:9EA3AD0C A:381A0273 Ack (145 bytes)

226 17:54:55.737   T      I -> R   S:9EA3AD9D A:381A0273 Ack (536 bytes)

226 17:54:55.889   T      I -> R   S:9EA3AFB5 A:381A0273 Ack (536 bytes)

226 17:54:56.041   T      I -> R   S:9EA3B1CD A:381A0273 Ack (536 bytes)

226 17:55:06.222    R     I <- R   S:381A0273 A:9EA3AD0C Ack

226 17:55:23.724   T      I -> R   S:9EA3AD0C A:381A0273 Ack (536 bytes)

226 17:55:29.483    R     I <- R   S:381A0273 A:9EA3AF24 Ack

226 17:55:29.504   T      I -> R   S:9EA3AF24 A:381A0273 Ack (536 bytes)

226 17:55:29.657   T      I -> R   S:9EA3B13C A:381A0273 Ack (536 bytes)

226 17:55:39.401    R     I <- R   S:381A0273 A:9EA3B13C Ack

226 17:55:39.421    R     I <- R   S:381A0273 A:9EA3B3E5 Ack

226 17:55:39.422   T      I -> R   S:9EA3B354 A:381A0273 Ack (145 bytes)
```

```
226 17:55:39.473   T     I -> R   S:9EA3B3E5 A:381A0273 Ack (536 bytes)
226 17:55:39.626   T     I -> R   S:9EA3B5FD A:381A0273 Ack (536 bytes)
226 17:55:39.779   T     I -> R   S:9EA3B815 A:381A0273 Ack Psh (536 bytes)
226 17:55:46.820    R    I <- R   S:381A0273 A:9EA3B3E5 Ack
226 17:55:46.836    R    I <- R   S:381A0273 A:9EA3B5FD Ack
226 17:55:46.854   T     I -> R   S:9EA3BA2D A:381A0273 Ack Psh (536 bytes)
226 17:55:54.657    R    I <- R   S:381A0273 A:9EA3B815 Ack
226 17:55:54.673    R    I <- R   S:381A0273 A:9EA3B815 Ack
226 17:55:54.693    R    I <- R   S:381A0273 A:9EA3B815 Ack
226 17:55:54.709    R    I <- R   S:381A0273 A:9EA3B815 Ack
226 17:55:54.673   T     I -> R   S:9EA3BC45 A:381A0273 Ack Psh (536 bytes)
226 17:56:01.252    R    I <- R   S:381A0273 A:9EA3BA2D Ack
226 17:56:01.268    R    I <- R   S:381A0273 A:9EA3BC45 Ack
226 17:56:06.830    R    I <- R   S:381A0273 A:9EA3BE5D Ack
226 17:56:06.848   T     I -> R   S:9EA3BE5D A:381A0273 Ack (536 bytes)
226 17:56:06.999   T     I -> R   S:9EA3C075 A:381A0273 Ack Psh (536 bytes)
226 17:56:16.368    R    I <- R   S:381A0273 A:9EA3C075 Ack
226 17:56:16.384    R    I <- R   S:381A0273 A:9EA3C28D Ack
226 17:56:16.385   T     I -> R   S:9EA3C28D A:381A0273 Ack (536 bytes)
226 17:56:16.538   T     I -> R   S:9EA3C4A5 A:381A0273 Ack Psh (536 bytes)
226 17:56:16.689   T     I -> R   S:9EA3C6BD A:381A0273 Ack Psh (536 bytes)
226 17:56:26.979    R    I <- R   S:381A0273 A:9EA3C4A5 Ack
226 17:56:26.995    R    I <- R   S:381A0273 A:9EA3C6BD Ack
226 17:56:26.996   T     I -> R   S:9EA3C8D5 A:381A0273 Ack Psh (536 bytes)
226 17:56:27.149   T     I -> R   S:9EA3CAED A:381A0273 Ack Psh (536 bytes)
226 17:56:35.229    R    I <- R   S:381A0273 A:9EA3C8D5 Ack
226 17:56:35.245    R    I <- R   S:381A0273 A:9EA3CAED Ack
226 17:56:35.261    R    I <- R   S:381A0273 A:9EA3CD05 Ack
226 17:56:35.246   T     I -> R   S:9EA3CD05 A:381A0273 Ack (536 bytes)
226 17:56:35.397   T     I -> R   S:9EA3CF1D A:381A0273 Ack Psh (536 bytes)
226 17:56:35.549   T     I -> R   S:9EA3D135 A:381A0273 Ack Psh (536 bytes)
226 17:56:35.700   T     I -> R   S:9EA3D34D A:381A0273 Ack Psh (536 bytes)
```

```
226 17:57:00.236   T      I -> R   S:9EA3CD05 A:381A0273 Ack (536 bytes)
226 17:57:36.628    R     I <- R   S:381A0273 A:9EA3CF1D Ack
226 17:57:36.644    R     I <- R   S:381A0273 A:9EA3D135 Ack
226 17:57:36.660    R     I <- R   S:381A0273 A:9EA3D34D Ack
226 17:57:36.645   T      I -> R   S:9EA3CF1D A:381A0273 Ack (536 bytes)
226 17:57:36.796   T      I -> R   S:9EA3D135 A:381A0273 Ack (536 bytes)
226 17:57:36.948   T      I -> R   S:9EA3D34D A:381A0273 Ack (536 bytes)
226 17:57:37.100   T      I -> R   S:9EA3D565 A:381A0273 Ack Psh (536 bytes)
226 17:57:37.253   T      I -> R   S:9EA3D77D A:381A0273 Ack Psh (536 bytes)
226 17:57:44.231    R     I <- R   S:381A0273 A:9EA3D565 Ack
226 17:57:44.247    R     I <- R   S:381A0273 A:9EA3D565 Ack
226 17:57:44.273    R     I <- R   S:381A0273 A:9EA3D565 Ack
226 17:57:44.248   T      I -> R   S:9EA3D995 A:381A0273 Ack Psh (536 bytes)
226 17:57:54.031    R     I <- R   S:381A0273 A:9EA3D565 Ack
226 17:57:54.047    R     I <- R   S:381A0273 A:9EA3D77D Ack
226 17:57:54.064    R     I <- R   S:381A0273 A:9EA3D995 Ack
226 17:57:54.080    R     I <- R   S:381A0273 A:9EA3DBAD Ack
226 17:57:54.065   T      I -> R   S:9EA3DBAD A:381A0273 Ack Psh (536 bytes)
226 17:57:54.223   T      I -> R   S:9EA3DDC5 A:381A0273 Ack (536 bytes)
226 17:57:54.381   T      I -> R   S:9EA3DFDD A:381A0273 Ack Psh (536 bytes)
226 17:57:54.534   T      I -> R   S:9EA3E1F5 A:381A0273 Ack Psh (536 bytes)
226 17:58:05.334    R     I <- R   S:381A0273 A:9EA3DDC5 Ack
226 17:58:05.349    R     I <- R   S:381A0273 A:9EA3DFDD Ack
226 17:58:05.365    R     I <- R   S:381A0273 A:9EA3DFDD Ack
226 17:58:05.381    R     I <- R   S:381A0273 A:9EA3E1F5 Ack
226 17:58:05.350   T      I -> R   S:9EA3E40D A:381A0273 Ack Psh (536 bytes)
226 17:58:05.504   T      I -> R   S:9EA3E625 A:381A0273 Ack Psh (356 bytes)
226 17:58:11.254    R     I <- R   S:381A0273 A:9EA3E40D Ack
226 17:58:11.270    R     I <- R   S:381A0273 A:9EA3E625 Ack
226 17:58:16.884    R     I <- R   S:381A0273 A:9EA3E789 Ack
226 18:02:31.143    R     I <- R   S:381A0273 A:9EA3E789 Ack Psh ( 93 bytes)
226 18:02:31.186   T      I -> R   S:9EA3E789 A:381A02D0 Ack
```

```
226 18:02:31.186    R     I <- R   S:381A02D0 A:9EA3E789 Ack Fin
226 18:02:31.198   T      I -> R   S:9EA3E789 A:381A02D0 Ack Fin
226 18:02:31.211   T      I -> R   S:9EA3E78A A:381A02D1 Ack
226 18:02:38.350    R     I <- R   S:381A02D1 A:9EA3E78A Ack
```

Distribution

US Military Acdmy
Mathematical Sci Ctr of Excellence
ATTN LTC T Rugenstein
Thayer Hall Rm 226C
West Point NY 10996-1786

TECOM
ATTN AMSTE-CL
Aberdeen Proving Ground MD 21005-5057

US Army CECOM
ATTN AMSEL-RD-ST-SP P VanSyckle
Ft Monmouth NJ 07703-5203

Director
US Army Rsrch Lab
ATTN AMSRL-RO-EN W D Bach
PO Box 12211
Research Triangle Park NC 27709

USATC
ATTN STEAC-TE-AS H Nguyen
Aberdeen Proving Ground MD 21005

US Army Rsrch Lab
ATTN AMSRL-CI-CB H Nguyen
ATTN AMSRL-CI-CN B Nguyen
(5 copies)
ATTN AMSRL-CI-CN G Cirincione
ATTN AMSRL-CI-CN G Racine
ATTN AMSRL-CI-CN G Tran
ATTN AMSRL-CI-IS-R Mail & Records
Mgmt
ATTN AMSRL-CI-IS-T Techl Pub
(2 copies)
ATTN AMSRL-CI-OK-TL Techl Lib
(2 copies)
ATTN AMSRL-D D R Smith
ATTN AMSRL-D J M Miller
Adelphi MD 20783-1197

INTENTIONALLY LEFT BLANK.